Statistical Techniques in Robotics (16-831, S21) Lecture #24 (Wednesday, April 28)

Max-Margin IRL and Max-Margin Planning

Lecturer: Kris Kitani *Scribes: Ingrid Navarro, Zach Patterson*

1 Review

Reinforcement Learning is great, but it takes a very long time to learn because everything has to be learned from experience. To increase learning speed, inspired by childhood learning, we can learn by demonstration of a human (or other "expert"). This is called Imitation Learning, although it's also known as Learning from Demonstration, apprenticeship learning, behavior cloning, or inverse reinforcement learning. We can think of this class of learning as belonging to one of three types, shown in Table 1. Briefly, in Passive IL, the policy is learned from expert demonstrations alone. In Active, the agent learns from demonstrations but also is able to act in the environment (this is the most common currently). In Interactive, the agent can learn from demonstrations and act in the environment, but also can allow for an "oracle" to step in for a new demonstration - think self driving cars asking the human to navigate a particularly complex situation.

	Passive IL	**Active IL**	**Interactive IL**
Demonstrations $\mathcal{D}$	yes	yes	optional
Environment $\mathcal{E}$	no	yes	yes
Oracle π	no	no	yes
Dynamics $\mathcal{T}$	no	optional	optional
Reward $\mathcal{R}$	no	optional	optional

Here, we will focus on a type of Active IL called Inverse Reinforcement Learning (IRL). It's called this because we, in a sense, invert the RL problem by using sampled expert demos to find a reward function. This helps us understand *why* the expert takes a certain action rather than just blindly following a learned expert policy (which may not generalize well). So we can transfer this learned reward function to generate new policies in new environments. In the rest of these notes, we'll cover several IRL algorithms. For the rest of this review section, we'll briefly review two Linear Program IRL [1] algorithms leading into our full LP IRL for sampled trajectories discussion in the Summary section.

The first algorithm assumes finite state spaces, a known transition model, and a known complete policy (all of which are not very realistic). To generate an objective function, the goal of this algorithm is to maximize the difference between the cumulative rewards for the expert policy and that of the next best policy:

$$\max_{\mathbf{R}} \{ \sum_{s} Q^{\pi}(s, a^*) - \max_{a \neq a^*} Q^{\pi}(s, a) \} - \lambda ||\mathbf{R}||_1. \tag{1}$$

We added regularization to stabilize the optimization. Following the definition of Q, we can derive the objective function:

$$\arg\max_{\mathbf{R}}(\sum_{s}\min_{a}\{(\mathbf{P}_{a^*}(s)-\mathbf{P}_a(s))(\mathbf{I}_\gamma\mathbf{P}_{a^*}(s))^{-1}\mathbf{R}\}-\lambda||\mathbf{R}||_1. \tag{2}$$

where $\mathbf{P}$ is the MDP dynamics and $\mathbf{R}$ is the matrix of rewards. This is a linear program that can be solved by typical linear program algorithms. See previous notes for a full derivation.

To adapt this algorithm for higher dimensional state spaces, we can approximate the reward function with a linear function (or a deep neural network), where the linear function is a function of some features of state:

$$R(x;\theta)=\sum_n \theta_n f_n(x)=\boldsymbol{\theta}^\top\boldsymbol{\phi}(s). \tag{3}$$

These features can, for example, represent a predicted classification (car, human, bicycle, etc). This allows us to write the value function as a linear function of expected state features

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[\sum_{t=0}^{\infty}\gamma^t r(s^{(t)})|s^{(0)}=s] \\ &= \mathbb{E}[\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\theta}^\top\boldsymbol{\phi}(s^{(t)})|s^{(0)}=s] \\ &= \boldsymbol{\theta}^\top\cdot\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\phi}(s^{(t)})|s^{(0)}=s] \\ V^\pi(s) &= \boldsymbol{\theta}^\top\boldsymbol{\mu}^{\boldsymbol{\pi}}(s) \end{aligned} \tag{4}$$

where $\boldsymbol{\mu}$ is the expected feature count. So, the concept for our objective function is similar here; we want to maximize the difference in our rewards for the expert policy and the next best policy, but this time it's over expected rewards so we'll maximize the difference between our value functions:

$$\begin{aligned} &\max_\theta \sum_{s\in S_0}\min a(\mathbb{E}_{p(s,a^*)}[V^\pi(s)]-\mathbb{E}_{p(s,a)}[V^\pi(s)]) \\ &\text{s.t.}|\theta_d|\le 1 \quad \forall d \end{aligned} \tag{5}$$

Remember, the value function contains the reward approximation function so we're still maximizing the difference between rewards. Additionally, we added the minimization so we're only comparing to the next best policy (so we don't have to compare to every policy). Finally, we sample from states S_0 instead of the full set of states - because the full set of states is infinite. We add the constraint to make sure that the solution does not explode.

2 Summary

2.1 Linear Program IRL

The previous objective function is derived with the assumption that we know the transition model and the policy, but this is usually not true. We usually only have expert trajectories, from which we'll need to estimate the value function. We want to use

$$V^{\pi}(s) = \boldsymbol{\theta}^{\top} \boldsymbol{\mu}^{\boldsymbol{\pi}}(s) \tag{6}$$

to estimate the value function, but we can't compute the expected feature count without the policy and transition dynamics. So, how do we obtain this information? There are multiple methods, but here we will use Monte Carlo Prediction, wherein we treat M expert trajectories as random Monte Carlo samples.

$$\begin{aligned} \boldsymbol{\mu}^{\boldsymbol{\pi}}(s) &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(s^{(t)})] \\ &\approx \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{T_m} \gamma^t \boldsymbol{\phi}(s_m^{(t)}). \end{aligned} \tag{7}$$

So now, as before, we want to maximize the difference between the value function estimate under some optimal policy and the estimate under some other policy,

$$\max_{\theta} \{\hat{V}^{\pi^*}(s) - \hat{V}^{\pi}(s)\}. \tag{8}$$

However, this is not quite complete yet. For one, there is no bound on the reward parameter θ so the solution will go to infinity. The other, more subtle problem is that we don't know where to get the 'other' policy. For the first issue, we simply regularize θ. For the second issue, we can use Reinforcement Learning.

The following is the regularized objective function for the linear program that we will use to solve this IRL problem:

$$\begin{aligned} \hat{\boldsymbol{\theta}} = \arg\max_{\theta} &\{\sum_{n} \hat{V}^{\pi^*}(s) - \hat{V}^{\pi_n}(s)\}. \\ \text{s.t.} &|\theta_d| \leq 1 \quad \forall d. \end{aligned} \tag{9}$$

With this objective function in mind, we can write the LP-IRL algorithm (see 2.1).

Algorithm 1 LP-IRL

1: **function** LP-IRL(π_0)
2: $\mathcal{D}^* = \{\zeta_j^*\}_{j=1}^J$ where $\zeta_j^* \sim \mathcal{E}|\pi^*$
3: **for** $t = 1, \ldots, T$ **do**
4: $\hat{V}^{\pi^*} = \text{MC-PREDICTION}(\mathcal{D}^*, r_{\theta_t})$
5: $\mathcal{D} = \{\zeta_k\}_{k=1}^K$ where $\zeta \sim \mathcal{E}|\pi_{t-1}$
6: $\hat{V}^{\pi_t}, \pi_t = \text{MC-CONTROL}(\mathcal{D}, r_{\theta_t})$
7: $\theta^{t+1} = \text{LINEARPROG}(\hat{V}^{\pi^*}, \{\hat{V}^{\pi_n}\}_{n=1}^t)$
8: **end for**
9: **return** θ
10: **end function**

Going line by line, line 2 is sampling from the expert demonstrations, line 4 approximates the value function from the expert policy using Monte Carlo, line 5 samples trajectories under the current policy and then line 6 uses Monte Carlo Control to approximate the value function and update the policy - this is the Reinforcement Learning subroutine - and line 7 runs a linear program over the previously specified objective function (Equation 2.1) to optimize reward parameters.

2.2 Matrix Game IRL

In this section we will derive a game theoretic approach to the IRL problem following [2]. In this paradigm, we'll use what is called the policy value to derive our objective function. In the past, when we have written the value function, we have taken the expectation over the MDP dynamics and policy starting from a given initial state s. In the following, we will take the expectation over an initial probability distribution of starting states. We will adopt the following shorthand to denote this policy value $\mathbb{E}[V^\pi(s)] \rightarrow V(\pi)$. Now, similar to before, we can derive an approximation of the policy value in terms of expected policy features (the expected future feature counts by following the policy over all possible starting points):

$$\begin{aligned} V(\pi) &= \mathbb{E}_p[\sum_{t=0} \infty \gamma^t r(s_t)] \\ &= \mathbb{E}_p[\sum_{t=0} \infty \gamma^t \theta \phi(s_t)] \\ &= \boldsymbol{\theta} \cdot \mathbb{E}_p[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)] \\ &= \boldsymbol{\theta} \cdot \boldsymbol{\mu}(\boldsymbol{\pi}). \end{aligned} \tag{10}$$

Now we need to derive our objective function. As before, we want the value of our expert policy to be better than any other policy;

$$V(\pi^*) \geq V(\pi). \tag{11}$$

We can rewrite the above by substituting our previously derived expression in expected policy features:

$$\boldsymbol{\theta} \cdot \boldsymbol{\mu}(\boldsymbol{\pi})^* \geq \boldsymbol{\theta} \cdot \boldsymbol{\mu}(\boldsymbol{\pi}). \tag{12}$$

Now, we'd like to have the largest possible difference between these terms, and we'd like to have that relation between the expert policy (now represented by π_E and the best 'other' policy. Similar to before, we can write this in terms of an optimization problem, but this time we flip signs and write it as a minimum over the whole function with a maximum expected value of the next best policy:

$$\min_{\theta}\{\max_{\pi} \boldsymbol{\theta} \cdot \mu(\pi) - \boldsymbol{\theta} \cdot \mu(\pi_E)\}. \tag{13}$$

We can pull the parameter vector $\boldsymbol{\theta}$ out to get the following simplified expression,

$$\min_{\theta}\{\max_{\pi} \boldsymbol{\theta}^\top (\mu(\pi) - \mu(\pi_E))\}. \tag{14}$$

That vector is a 1xN vector of the feature parameters. By adjusting our notation, we can understand $(\mu(\pi) - \mu(\pi_E))$ as a matrix multiplication between a NxM matrix, $\boldsymbol{G}$, and a Mx1 vector, $\boldsymbol{\psi}$;

$$\min_{\theta}\{\max_{\pi} \boldsymbol{\theta}^\top \boldsymbol{G}\boldsymbol{\psi}\}. \tag{15}$$

This notation reveals that IRL is a two player game (see [3]) where $\boldsymbol{G}$ specifies the game, $\boldsymbol{\theta}$ specifies the row player, and $\boldsymbol{\psi}$ specifies a column player (adversarial). N is the number of features and M is the number of deterministic policies.

Using the knowledge that IRL can be understood as a two player game, we can adapt well known algorithms developed to solve such games to solve this problem, such as the Multiplicative Weights Algorithm.

Algorithm 2 Multiplicative Weights Matrix Game

1: **function** MW-MatrixGame(β)
2: $\quad \boldsymbol{w}^{(0)} \leftarrow \{w_r^{(0)} = 1\}$
3: $\quad$ **for** $t = 1, \ldots, T$ **do**
4: $\quad\quad R^{(t)} = \frac{\boldsymbol{w}^{(t-1)}}{\sum_r w_r^{(t-1)}}$
5: $\quad\quad$ Receive($\boldsymbol{l}^{(t)} = \mathrm{G}(\cdot, \boldsymbol{c}^{(t)})$)
6: $\quad\quad w_r^{(t)} = w_r^{(t-1)} \beta^{l_r^{(t)}} \quad \forall r$
7: $\quad$ **end for**
8: **return** $\boldsymbol{\theta}$
9: **end function**

Walking through this algorithm, line 4 designates the row player strategy, line 5 shows the game matrix and the column player strategy, and line 6 updates the row player strategy. When we walk through the Multiplicative Weights Algorithm (see 2.2, we can see that this is really online

exponentiated gradient descent (line 6), which brings the class rather full circle. Using the MW Algorithm, we can write a Multiplicative Weights IRL Algorithm.

Algorithm 3 Multiplicative Weights IRL

1: **function** MW-IRL($\hat{\pi}_E, \hat{\mu}_E, \beta$)
2: $\quad \boldsymbol{\theta} \leftarrow 0$
3: $\quad$ **for** $t = 1, \ldots, T$ **do**
4: $\quad\quad \theta_i = \frac{\theta_i}{\sum_{i'} \theta_{i'}} \quad \forall i$
5: $\quad\quad \hat{\pi}_\theta = \mathrm{RL}(r(\cdot; \boldsymbol{\theta}))$
6: $\quad\quad \hat{\mu} = \textsc{Estimate}()$
7: $\quad\quad \theta_i = \theta_i \cdot \exp\{\ln \beta \cdot G_i(\hat{\mu}_i)\} \quad \forall i$
8: $\quad$ **end for**
9: **return** $\boldsymbol{\theta}$
10: **end function**

For a derivation and in depth coverage of the above, please see [2]. Briefly and essentially, we update our weights (line 4), use RL over our reward parameter to calculate some optimal policy $\hat{\pi}$ (line 5), use the optimal policy to calculate the expected feature count $\hat{\mu}$ (line 6), before updating with exponentiated gradient descent. Lines 5 and 6 can be understood as a 'prediction step' where the actual game is played.

2.3 Max Margin IRL

The section that follows provides a summary of the Max-Margin IRL algorithm which was introduced in [4] as a quadratic programming problem for IRL. Max-Margin IRL is not guaranteed to recover a true reward function, however, it will find a policy that performs as well as the expert. Furthermore, the algorithm was shown to terminate in a small number of iterations.

The setting for this algorithm is framed as a MDP where a reward function is not available, rather, the agent observes expert behavior from which will attempt to recover the unknown reward. This setting assumes that such reward can be expressed as linear combination of known "features".

More formally (and following the notation introduced in the previous sections), we can express the optimization problem for Max-Margin IRL as:

$$\begin{aligned} \max_{\theta} \; & t \quad \text{where } t \text{ is a margin variable} \\ & \text{s.t. } \boldsymbol{\theta}^T \boldsymbol{\mu}(\pi^*) \geq \boldsymbol{\theta}^T \boldsymbol{\mu}(\pi) + t \quad \forall \pi \end{aligned} \tag{16}$$

i.e., we want to find a reward function for which the expert policy does better by a margin.

At this point there are two main problems:

1. There is no bound on the reward parameters (parameters can explode!)
2. We cannot compare against all possible policies

Now, to address the former issue, we can regularize the parameters of the reward function. For the latter, we can leverage reinforcement learning to find n valid policies to compare against.

Through this approach, the regularized objective thus becomes:

$$\begin{aligned} &\max_{\theta} \; t \\ &\quad \text{s.t. } \boldsymbol{\theta}^T \boldsymbol{\mu}(\pi^*) \geq \boldsymbol{\theta}^T \boldsymbol{\mu}(\pi) + t \quad \forall \pi \\ &\quad\quad ||\boldsymbol{\theta}||_2 \leq 1 \end{aligned} \tag{17}$$

We can further re-write the objective as a minimization as follows:

$$\begin{aligned} &\min_{\theta, t} \quad \lambda ||\boldsymbol{\theta}||_2 - t \\ &\quad \text{s.t.} \quad \boldsymbol{\theta}^T (\boldsymbol{\mu}(\pi^*) - \boldsymbol{\mu}(\pi)) \geq t \quad \forall \pi_n \in \Pi \end{aligned} \tag{18}$$

We can observe that above minimization objective follows a similar form to the objective of a SVM max-margin classifier:

$$\begin{aligned} &\min_{w, \xi} ||\boldsymbol{w}||_2 + C \sum_i \xi_i \\ &\quad \text{s.t.} \quad y_i (\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i \end{aligned} \tag{19}$$

This shows that IRL can be framed as a max-margin classifier. In this class, such types of problems have been solved using online gradient descent. Nonetheless, Max-Margin IRL can also be solved using quadratic programming, and Algorithm 2.3 showcases the procedure to do so.

Algorithm 4 Max-Margin IRL

1: **function** MM-IRL($\mu(\pi^*)$)
2: **for** $i = 1, \ldots n$ **do**
3: $\hat{\pi} = \text{argmax}_{\pi} \boldsymbol{\theta}^T \boldsymbol{\mu}(\pi)$
4: $\boldsymbol{\mu}(\hat{\pi}) = \mathbb{E}_{p_0, \mathcal{T}, \hat{\pi}} \big[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \big]$
5: $\boldsymbol{\theta} = \textsc{QuadProg}(\lambda, \boldsymbol{\mu}(\pi^*), \{\boldsymbol{\mu}(\hat{\pi})\}_{n=1}^{i})$
6: **end for**
7: **return** $\boldsymbol{\theta}$
8: **end function**

Line by line, the algorithm goes as follows: line 2 is iterates from $i = 1$ to $i = n$, where n represents the number of policies that we are using. Line 3 estimates an optimal policy using RL with the current estimate of the reward parameters $\boldsymbol{\theta}$. Line 4 computes the expected policy value, and line 5: updates the parameters using the computed features. Note that line 5 can be replaced to use online gradient descent instead of quadratic programming to perform the parameter update. Finally, line 7 returns $\boldsymbol{\theta}$ once we have gone through all of the available policies.

2.4 Structured Output Max Margin IRL

This section discusses Structured Output Max-Margin IRL, also known as Max-Margin Planning (MMP), an algorithm that learns to plan by solving a Structured Maximum Margin prediction problem over a space of policies.

Max-Margin IRL, introduced in the previous section, differs from MMP in that the latter allows demonstration of policies from more than a single MDP: they allow demonstrations from multiple feature maps, and with different start and goal states [5].

First, observe that the minimization objective derived in Eq. 18 for the Max-Margin IRL algorithm uses a margin of 1. In contrast, MMP uses the notion of a variable margin, which depends on a distance between policies. This leads to the idea of Structured Prediction Max-Margin:

$$\begin{aligned} \min_{\theta,\xi} \quad & \lambda||\boldsymbol{\theta}||_2 + \sum_i \xi_i \\ \text{s.t.} \quad & \boldsymbol{\theta}^T(\boldsymbol{\mu}(\pi^*) - \boldsymbol{\mu}(\pi)) \geq l(\pi^*, \pi_i) - \xi_i \quad \forall i \end{aligned} \tag{20}$$

In the above equation, $l(\pi^*, \pi)$ is the variable loss (or margin). The intuition behind it is that if two policies are significantly different, then margin should large. One example of a variable loss would be to sample trajectories from different policies and measuring the distance between them.

The simplified version of the objective function for MMP algorithm can be also expressed as in Eq. 21 (we refer the reader to the original paper for a full derivation):

$$\min_{\theta} \frac{1}{2}||\boldsymbol{\theta}||_2 + \lambda \sum_d \left\{\boldsymbol{\theta}^T F_d \eta + l_d^T \eta - \boldsymbol{\theta}^T F_d \eta_d\right\} \tag{21}$$

where $l(\pi^*, \pi_d) = l_d^T \eta^*$ represents the structural loss (or variable margin) with $\eta(a, s) \in \mathbb{R}^{|S||A|}$ as an occupancy measure that counts the number of times a state-action pair has been visited. Note that the expression above uses an alternative representation of the policy value given by:

$$V(\pi) = \boldsymbol{\theta}^T F \eta_\pi \tag{22}$$

where $F \in \mathbb{R}^{N \times |S||A|}$ is a feature map for each state-action pair, and $\mu = F\eta$ is the expected feature count.

The final objective for MMP is given by:

$$\min_{\theta} \frac{1}{2}||\boldsymbol{\theta}||_2^2 + \frac{\lambda}{D} \sum_d \beta_d \big\{ \max_{\eta} (\boldsymbol{\theta}^T F_d + l_d^T)\eta - \boldsymbol{\theta}^T F_d \eta_d \big\} \tag{23}$$

Note that the objective is similar to the SVM loss:

$$\min_{w} \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{1}{M} \sum_{m=1}^{M} \max\{0, 1 - y_m \boldsymbol{w}^T \boldsymbol{x}_m)\} \tag{24}$$

Thus, we can use online (sub)-gradient descent to solve this problem using the sub-gradient update equation below:

$$\boldsymbol{g} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \boldsymbol{\theta} + \frac{\lambda}{D} \sum_d \beta_d F_d (\eta^* - \eta_d) \tag{25}$$

Finally, in class we covered two versions of the Max-Marging Planning: MMP-batch shown in Algorithm 2.4, and MMP-online in Algorithm 2.4. We describe the batch version of the algorithm which performs gradient descent over all trajectories, whereas the online version differs in that

one trajectory is taken at at time. In line 4, the policy π_d is obtained using RL, using all of the trajectories d. Line 5 computes the visitation η_d count of the state-action pairs. Then, the sub-gradient is computed, and then used to update the parameters. The whole process is repeated for T-steps, and afterwards the function returns the final parameters.

Algorithm 5 Max-Margin Planning-Batch

1: **function** MM-PLANNING-BATCH($\{F_d, \eta_d, l_d\}_{d=1}^{D}, \lambda, \alpha, T$)
2: $\quad \boldsymbol{\theta} \leftarrow 0$
3: $\quad$ **for** $i = 1, \ldots, T$ **do**
4: $\quad\quad \pi_d = \text{RL}(\boldsymbol{\theta}^T F_d + l_d^T) \quad \forall d$
5: $\quad\quad \eta_d = \textsc{CountVisitation}((\pi_d)) \quad \forall d$
6: $\quad\quad g = \textsc{ComputeSubgrad}((\boldsymbol{\theta}, F_d, l_d, \eta_d)) \quad \forall d$
7: $\quad\quad \boldsymbol{\theta}_n = \boldsymbol{\theta}_n - \alpha_t \cdot g_n \quad \forall n$
8: $\quad$ **end for**
9: **return** $\boldsymbol{\theta}$
10: **end function**

Algorithm 6 Max-Margin Planning-Online

1: **function** MM-PLANNING-ONLINE($\{F_d, \eta_d, l_d\}_{d=1}^{D}, \lambda, \alpha, T$)
2: $\quad \boldsymbol{\theta} \leftarrow 0$
3: $\quad$ **for** $i = 1, \ldots, T$ **do**
4: $\quad\quad \pi_d = \text{RL}(\boldsymbol{\theta}^T F_d + l_d^T)$
5: $\quad\quad \eta_d = \textsc{CountVisitation}((\pi_d))$
6: $\quad\quad g = \textsc{ComputeSubgrad}((\boldsymbol{\theta}, F_d, l_d, \eta_d))$
7: $\quad\quad \boldsymbol{\theta}_n = \boldsymbol{\theta}_n - \alpha_t \cdot g_n \quad \forall n$
8: $\quad$ **end for**
9: **return** $\boldsymbol{\theta}$
10: **end function**

References

[1] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 663–670. Morgan Kaufmann, 2000.

[2] Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1449–1456. Curran Associates, Inc., 2007.

[3] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007.

[4] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In Carla E. Brodley, editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[5] Nathan D. Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. In William W. Cohen and Andrew W. Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 729–736. ACM, 2006.